Run, skeleton, run!
NIPS RL 2017 3rd place

Mikhail Pavlov, Sergey Kolesnikov
Reason8.ai

Presentation course

- About NIPS RL 2017 competition
- Simulator and environment
  - bugs & hacks
- Competition moments
- Our model
  - Method
  - Improvements
  - Comparing to baselines
- Best practices
- Agent policies examples
- Code, contacts and questions

NIPS RL 2017: Learning To Run

By: Stanford Neuromuscular Biomechanics Laboratory

Prizes:

1. NVIDIA DGX Station™
2. NVIDIA Titan Xp
3. **NVIDIA Titan Xp**

Additional rules:

- **NO** external datasets usage
- Rules may change

NIPS 2017

Timeline & Tasks

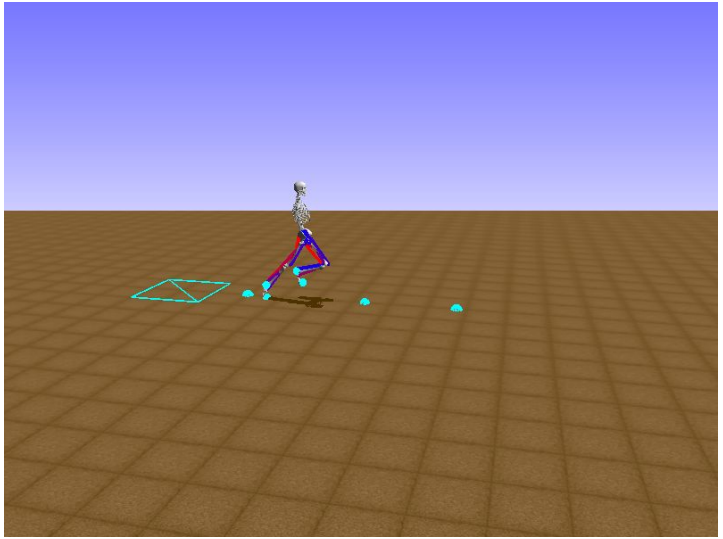
Round 1: submit > 15 meters for 3 obstacles

July → ~~October~~ →...→November 4

Round 2: submit docker image for 10 obstacles

October 20 → November ~~10~~ 13

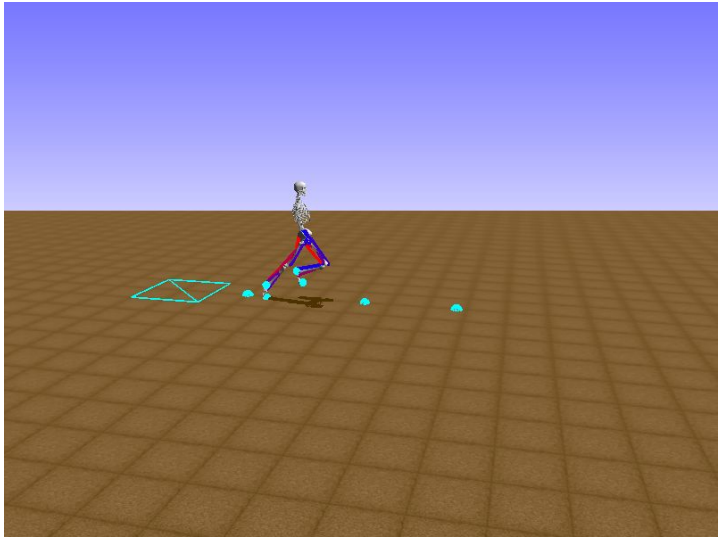
OpenSim environment



OpenSim screenshot that demonstrates the agent

parameters	description
state (a_t)	$\mathbb{R}^{41}$, coordinates and velocities of various body parts and obstacle locations. All (x, y) coordinates are absolute. To improve generalization of our controller and use data more efficiently, we modified the original version of environment making all x coordinates relative to the x coordinate of pelvis.
action (a_t)	$\mathbb{R}^{18}$, muscles activations, 9 per leg, each in $[0, 1]$ range.
reward	$\mathbb{R}$, change in x coordinate of pelvis plus a small penalty for using ligament forces.
terminal state	agent falls (pelvis $x < 0.65$) or 1000 steps in environment
stochasticity	• random strength of the psoas muscles • random location and size of obstacles

OpenSim environment



OpenSim screenshot that demonstrates the agent

~1600 times slower that MoJuCo/Roboschool

parameters	description
state (a_t)	$\mathbb{R}^{41}$, coordinates and velocities of various body parts and obstacle locations. All (x, y) coordinates are absolute. To improve generalization of our controller and use data more efficiently, we modified the original version of environment making all x coordinates relative to the x coordinate of pelvis.
action (a_t)	$\mathbb{R}^{18}$, muscles activations, 9 per leg, each in $[0, 1]$ range.
reward	$\mathbb{R}$, change in x coordinate of pelvis plus a small penalty for using ligament forces.
terminal state	agent falls (pelvis $x < 0.65$) or 1000 steps in environment
stochasticity	• random strength of the psoas muscles • random location and size of obstacles

Main competition moments

- Wrong reward signal ¯_(ツ)_/¯
- State transform
- Flip state action
- Skip frames
- Environment speedup (2.5x)
  - compiled with reduced accuracy
  - **~1-3** step per second for Round 1
  - **~0.01-1** step per second for Round 2
  - better agent generalization
- Round 1 seed leak

Performance of some known algorithms for continuous control

1. **TRPO**

$$\begin{aligned} \text{maximize}_\theta \quad & E_{s\sim\rho_{\theta'},a\sim\pi_{\theta'}}\left[\frac{\pi_\theta(a|s)}{\pi'_\theta(a|s)}A_{\theta'}(s,a)\right] \\ \text{s.t.} \quad & E_{s\sim\rho_{\theta'}}[D_{\mathrm{KL}}(\pi_{\theta'}(\cdot|s)\|\pi_\theta(\cdot|s))] \le \delta_{\mathrm{KL}} \end{aligned}$$

2. **PPO**

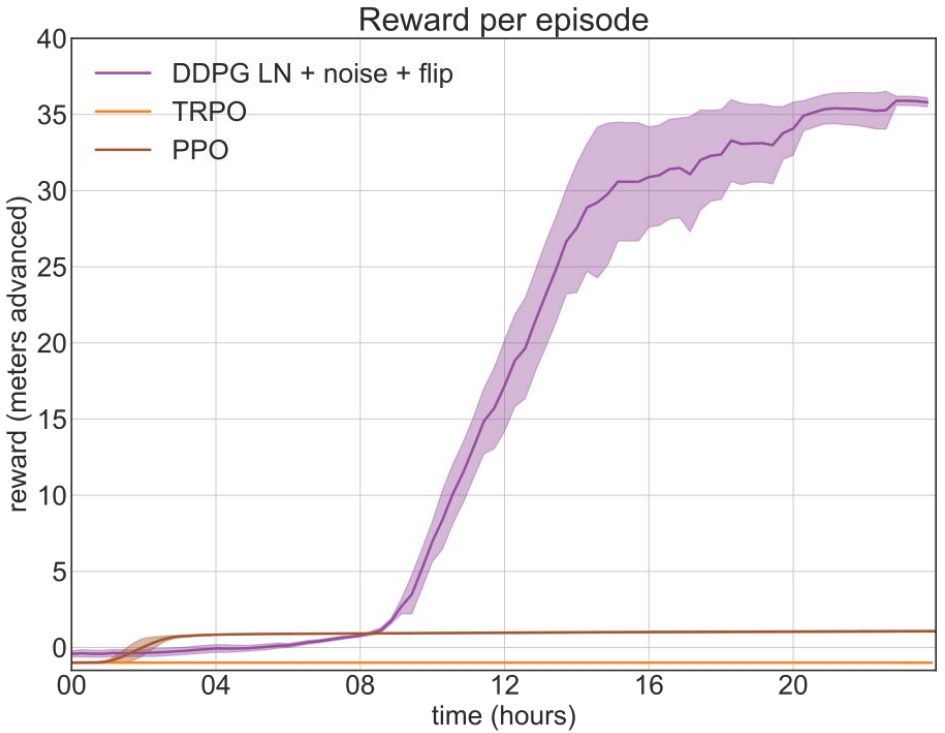
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \qquad \hat{A}_t = R_t - b_t$$

3. **DDPG**

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$



Performance of some known algorithms for continuous control

1. **TRPO**

$$\begin{aligned} &\text{maximize}_\theta \quad E_{s\sim\rho_{\theta'},a\sim\pi_{\theta'}}\left[\frac{\pi_\theta(a|s)}{\pi'_\theta(a|s)}A_{\theta'}(s,a)\right] \\ &\text{s.t.} \quad E_{s\sim\rho_{\theta'}}[D_{\mathrm{KL}}(\pi_{\theta'}(\cdot|s)\|\pi_\theta(\cdot|s))] \le \delta_{\mathrm{KL}} \end{aligned}$$

2. **PPO**

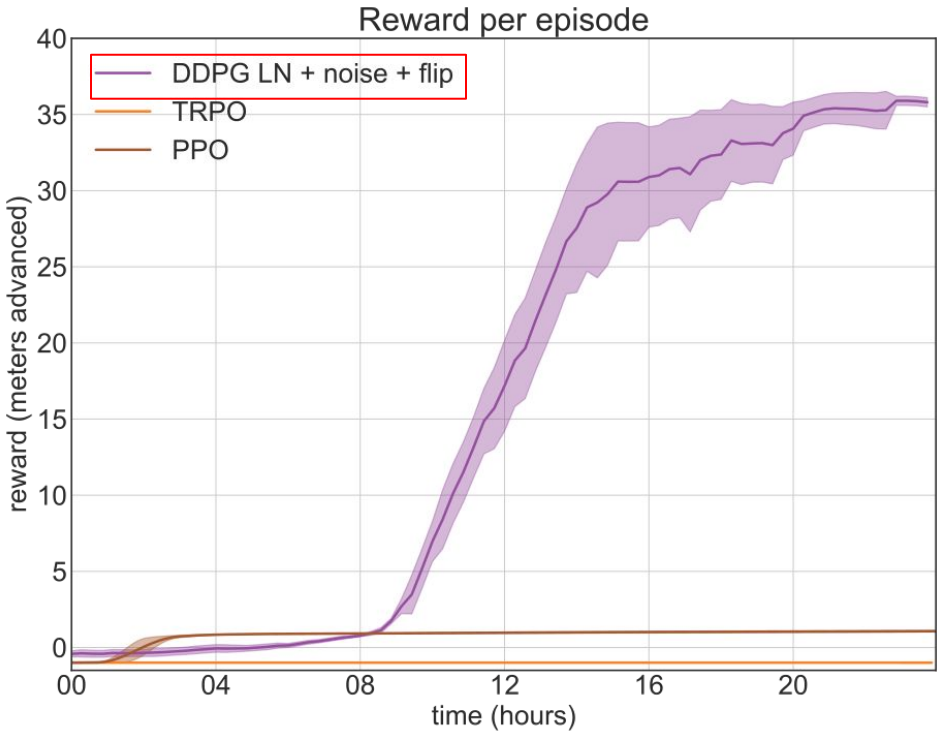
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \qquad \hat{A}_t = R_t - b_t$$

3. **DDPG**

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$



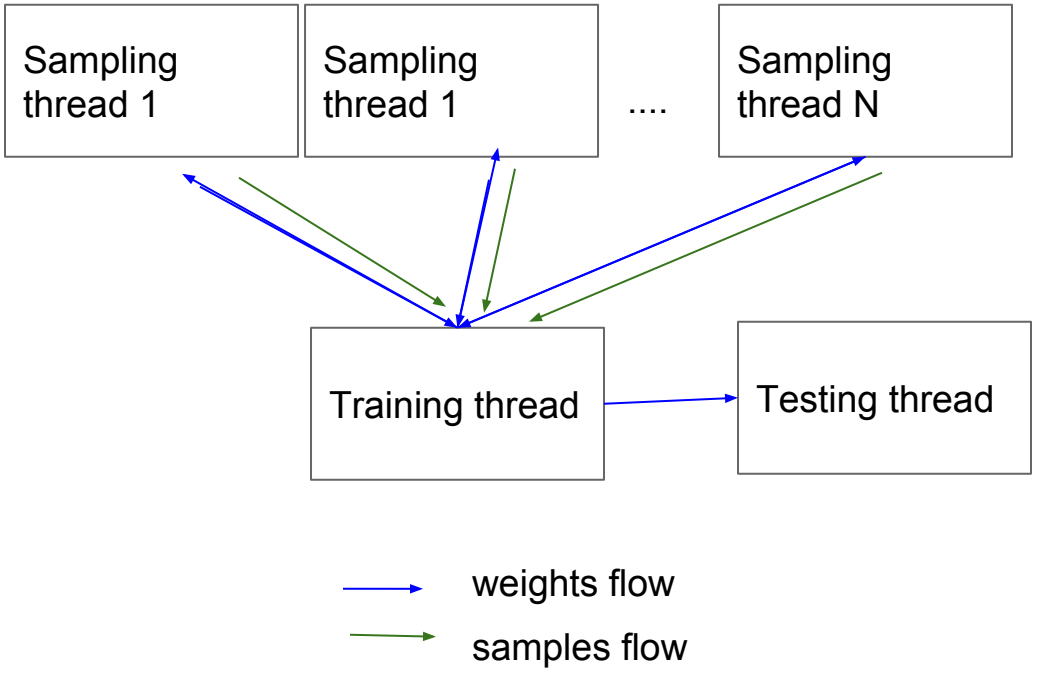
DDPG architecture and exploration

Key features:
- For sampling thread weights are fixed for entire episode, stale weights allows better exploration

- Sampling is done with actions (70% prob) and parameter noise (30%prob).

- Param noise is used only for experience replay sampling and And in case of high reward alse tested

- Separate thread for testing (as we need to evaluate weights without noise) test is done with 5 episodes and random seeds



Sample efficiency problem

Approach:

1) **Use DDPG with experience replay to use one sample many times**
2) **State have absolute x coordinate, make all x coordinates relative to pelvis coordinate**
3) **Flip action and states (increase sample size by 2, agent learns to use both legs)**

Other tricks

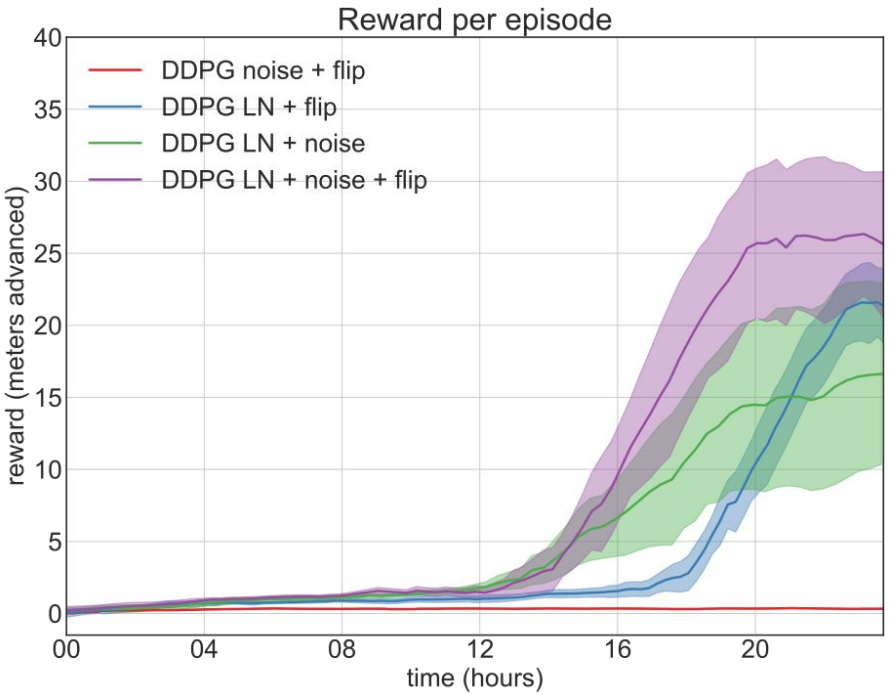
1. Repeat each action n-times (n = 5) during train, but 1 time during test (somehow it worked and helped to avoid obstacles in test)
2. Use reward scaling (multiply reward by 10)
3. Use layer normalization (according to articles models with layer norm works in wider range of reward scaling) and parameter noise.
4. Actor [64, 64, elu](first round)/[64, 64, 32, elu](second round); Critic [64, 32, tanh] (both rounds)
5. Gamma 0.9 (as we don't need to look far in future)

Last night

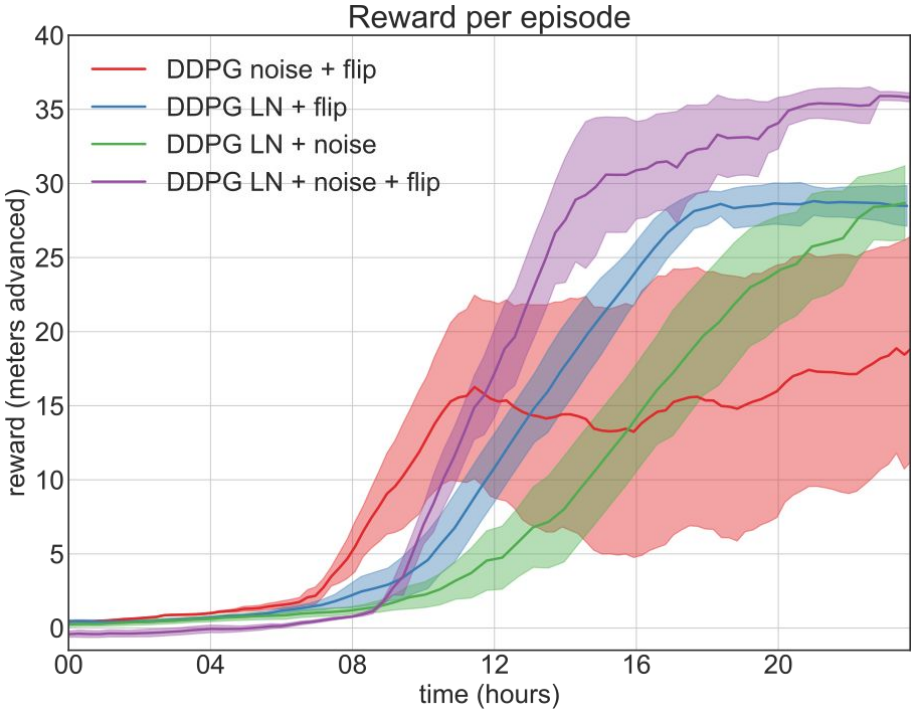
Fine-tuning ddpg with CEM algorithm (it learned better policy for some muscles strength)

Effect of improvements and number of threads

8 threads (6 sampling threads)



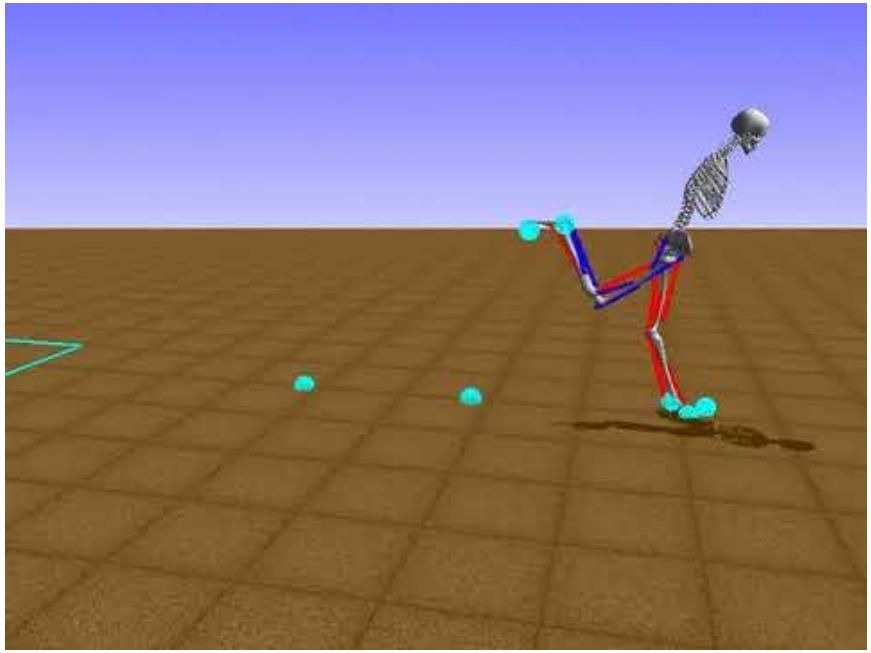
20 threads (18 sampling threads)

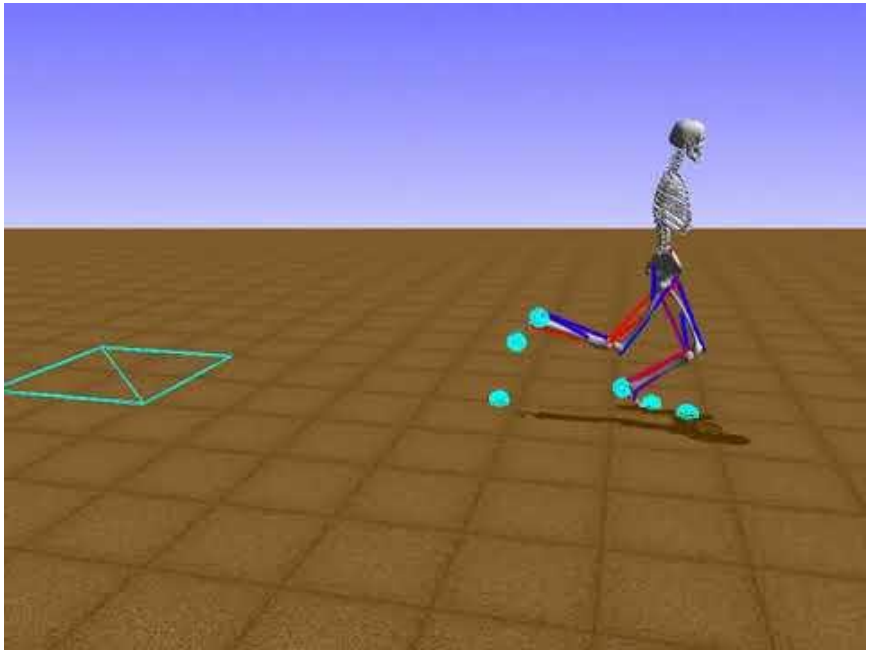


Summing it up
Best practices

- Carefully look at actions, states, reward and learned policy.
- Carefully read others participants hints on forum/gitter
- Change 1 thing at time (there is always temptation to change many things)
- Discuss your ideas with other people
- Do not despise reward scale and larger networks.
- Name your parameters correctly (cause strange bugs).
- Submit earlier.
- Use docker images for your future competition.

Run, skeleton, run!





Code & Contacts

Solution:

- LeaderBoard: https://www.crowdai.org/challenges/nips-2017-learning-to-run/leaderboards
- Theano version: https://github.com/fgvbrt/nips_rl
- PyTorch version: https://github.com/Scitator/Run-Skeleton-Run
- Fast OpenSim version: https://github.com/Scitator/opensim-core **Thanks to Ivan Sorokin**

CrowdAI - https://www.crowdai.org

Orim-RL - https://github.com/stanfordnmbl/osim-rl

OpenSim - https://github.com/opensim-org/opensim-core